CS/SE 4348: Operating Systems Concepts
Section 004
Programming Project 1

Instructor: Neeraj Mittal

Assigned on: Friday, February 7, 2025
Due date: Wednesday, February 27, 2025

This is an individual assignment. You are expected to write the code independently and submit your own work. However, you can freely discuss the implementation ideas with other students in the class. Copying or using work not your own will result in disciplinary action and the suspected incident will be referred to the Office of Community Standards and Conduct for investigation!

1 Project Description

Important: The goal of this project is to write a concurrent program for a UNIX/Linux based system using *processes* created via fork( ) system call.

Given an array A consisting of n numbers (may be negative), the prefix sum problem involves computing another array B of size n such that that i^{th} element in B is the sum of first i elements in A. Formally,

$$\forall i : 0 \leq i < n : B[i] \quad = \quad \sum_{j=0}^{i} A[j]$$

You have to implement the well-known *Hillis and Steele's concurrent algorithm* to solve the prefix-sum problem using multiple cores (see `https://en.wikipedia.org/wiki/Prefix\_sum`). The algorithm is presented here for your convenience.

The algorithm assumes that you have as many cores as the number of elements. You will have to adapt it to work for a given number of cores, and divide the work equally among the cores to the extent possible. Your program should accept four arguments:

(1) the number of elements in the input array, denoted by n,

(2) the number of cores, denoted by m,

(3) the input file that contains the elements in A, and

(4) the output file that will contain the elements in B.

Note that it is your responsibility to validate the arguments. For example, $n > 0$, $m > 0$, A.txt exists and contains at least n elements, and so on. In short, your program should not exhibit an undefined behavior if one or more input arguments is erroneous; if is not able to continue, it should terminate gracefully.

```
x[0] := the input array;
for p ← 1 to ⌈log2 n⌉ do
    for i ← 0 to n − 1 do in parallel
        if i < 2^(p−1) then
            x[p][i] := x[p − 1][i];
        else
            x[p][i] := x[p − 1][i − 2^(p−1)] + x[p − 1][i];
        end
    end
end
```

Algorithm 1: Hillis and Steele's concurrent prefix-sum algorithm.

```
wall[1..m] := {0, 0, ..., 0};                    // shared by all processes
// Code for process p_i
wall[i] := wall[i] + 1;
while ⟨∃j : wall[j] < wall[i]⟩ do
    ;                                                          // spin
end
```

Algorithm 2: A reusable barrier.

In your program, the main process should create m worker (child) processes to perform the work as required by modified Hillis and Steele's algorithm. The main process is responsible for reading the input array from the input file and writing the output array to the output file. Note that this assignment is about *concurrency using processes (created using fork( )) and not threads.*

Hillis and Steele's algorithm works *iteratively.* Each iteration generates a new array with the final iteration generating the output array. All worker processes must finish the current iteration before any worker process can start the next iteration. To achieve this synchronization, *you need to implement a barrier.* A barrier provides a single method arriveAndWait( ). When a process invokes the arriveAndWait( ) method, it is blocked until all other processes have also invoked the arriveAndWait( ) method. A simple algorithm for a reusable barrier has been included here for your convenience.

All processes should use shared memory created using shmget( ) system call to share all data structures, which includes the input array, the output array, all intermediate arrays as needed and any data structure used to implement a reusable barrier.

Important: Your program should use $O(n \log_2 n)$ space and run in $O(n/m \log_2 n + m \log_2 n)$ time to receive full credit.

You can write your program in C or C++. Name your program as `my-sum`. Also, *ensure that your program runs on one of the department machines cs1.utdallas.edu, cs2.utdallas.edu or giant.utdallas.edu; otherwise you will not get any credit.* Any deviation from the description would result in significant penalty.

2 Bonus Points

You can earn 10% bonus points if your program uses only $O(n)$ space. You can earn another 15% bonus points if your reusable barrier uses only $O(1)$ space.

3 Grading Criteria

As such, projects will be graded with these criteria in mind:

- Solutions must adequately address the problem at hand. Specifically:
  - The solution represents a good-faith attempt to actually address the requirements for the assignment.
  - The program complies and executes.
  - The program runs correctly.
- The solution constitutes a high quality product expected of a professional. Specifically:
  - The program is easy to read and to understand, that is, it is well commented. In addition, method and variable names are meaningful, all potentially confusing/complex code is well documented.
  - The general design of the program is clear and reasonable.
  - All procedure and function headers include comments explaining what the method is supposed to do (not how it does it) and the purpose of each formal parameter. Be as precise and careful as you can be.
  - The program is robust and handles important errors and exceptions properly.

4 Submission Information

You have to submit your project through eLearning. Along with all the *source* files, submit the following:

(i) A Makefile to compile the program, and

(ii) A README file that contains the names of all the team members, and the instructions for running the compiled program.

Points will be deducted if you fail to submit any of the above-mentioned files.